Looking Under the Hood of the IBM Blue Gene/Q Network

Dong Chen, Noel Eisley, Philip Heidelberger, Sameer Kumar, Amith Mamidala, Fabrizio Petrini, Robert Senger, Yutaka Sugawara, Robert Walkup
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
{chendong, naeisley, philiph, sameerk, amithr, fpetrin, rmsenger, ysugawa, walkup}@us.ibm.com

Anamitra Choudhury, Yogish Sabharwal, Swati Singhal
IBM India Research Lab
New Delhi, India
{anamchou, ysabharwal, swatisin}@in.ibm.com

Burkhard Steinmacher-Burow
IBM Deutschland Research & Development GmbH
71032 Böblingen, Germany
steinmac@de.ibm.com

Jeffrey J. Parker
IBM Systems &Technology Group
Systems Hardware Development
Rochester, MN 55901
jjparker@us.ibm.com

***Abstract*—This paper explores the performance and optimization of the IBM Blue Gene/Q (BG/Q) five dimensional torus network on up to 16K nodes. The BG/Q hardware supports multiple dynamic routing algorithms and different traffic patterns may require different algorithms to achieve best performance. Between 85% to 95% of peak network performance is achieved for all-to-all traffic, while over 85% of peak is obtained for challenging bisection pairings. A new software-controlled algorithm is developed for bisection traffic that selects which hardware algorithm to employ and achieves better performance than any individual hardware algorithm. The benefit of dynamic routing is shown for a highly non-uniform "transpose" traffic pattern. To evaluate memory and network performance, the HPCC Random Access benchmark was tuned for BG/Q and achieved 858 Giga Updates per Second (GUPS) on 16K nodes. To further accelerate message processing, the message libraries on BG/Q enable the offloading of messaging overhead onto dedicated communication threads. Several applications, including Algebraic Multigrid (AMG), exhibit from 3 to 20% gain using communication threads.**

Keywords- interconnection network; network performance; network routing; GUPS; Blue Gene;

I. INTRODUCTION

Blue Gene/Q (BG/Q) is the third generation of highly scalable, power efficient supercomputers in the IBM Blue Gene line, following Blue Gene/L [1] and Blue Gene/P [2]. A 96 rack, 20 petaflops, Blue Gene/Q system called Sequoia has been installed at the Lawrence Livermore National Laboratory, while a 48 rack configuration named Mira has been installed at the Argonne National Laboratory.

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

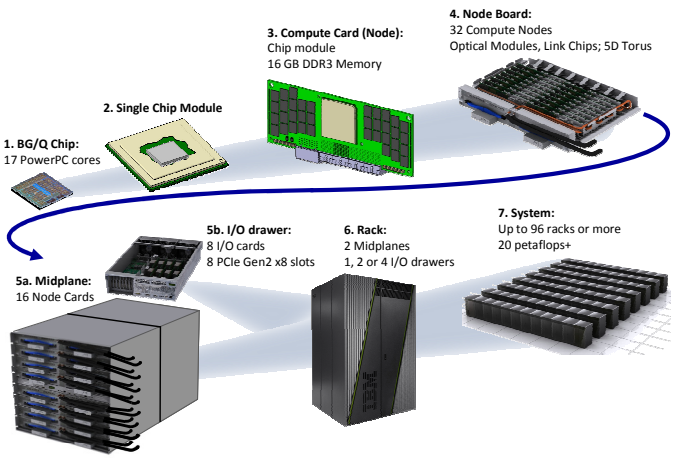

Figure 1. BG/Q dense packaging hierarchy for massive scale-out. ©2012 Springer Verlag. Reprinted, with permission, from [14].

BG/Q leverages a highly integrated System-on-a-Chip (SoC) design with custom on-die torus network and dense system-level packaging to provide a low-latency, low-power, high-bandwidth and cost efficient solution for massive scale-out installations. Design for scalability is especially important for large petaflop class machines where performance, density, and power are key inter-related system parameters. As shown in Figure 1, a BG/Q compute node consists of the SoC single-chip module with associated memory. 32 compute nodes are electrically interconnected to form a 2x2x2x2x2 grid on a node card. 16 node cards comprise a 512-node midplane and two midplanes stack vertically to form a 1024-node rack, with electrical links within midplanes and optical links between midplanes. Racks may also contain special I/O drawers with Gen-2 PCIe connectivity. The final BG/Q system scales to 96

racks and beyond. The racks are water cooled to permit maximum compute density.

An overview of BG/Q is given in [3]. The BG/Q SoC has 16 cores for user code, and a 17th core is reserved for use by the system software. Each core has four hardware threads. The 64-bit, in-order, PowerPC cores run at 1.6 GHz. A core can execute two instructions per cycle: a floating point instruction on one thread and an integer, branch, load or store on another thread. Each core has a four wide SIMD floating point engine capable of executing 8 floating point operations per cycle; the peak peformance of a node is 204.8 GFlops. A crossbar switch connects the cores to a 32 MB shared L2 cache, organized as 16 slices with 2 MB per slice. Detailed descriptions of the BG/Q five dimensional (5D) torus interconnection network and its associated DMA engine, called the Message Unit, which are integrated onto the same chip as the cores, are given in [4][5]. The Message Unit attaches to the cores and the memory system over the crossbar switch. Other notable uses of a torus interconnect in supercomputers include 3D Cray machines [6][7] and the 6D Fujitsu K computer [8]. Other scalable networks used in supercomputers today are clos [18] and dragon-fly [16] indirect networks, and all connected direct networks [17].

BG/Q was designed for scalability and power efficiency. Sequoia placed first on the June 2012 TOP500 list (http://www.top500.org) at 16.3 Petaflops, an efficiency of 81.1% of peak, and various configurations of BG/Q have ranked first on the four most recent Green500 lists (http://www.green500.org) for power efficiency (November 2010 to June 2012). Additionally, BG/Q ranked first on the November 2011 and June 2012 Graph 500 lists (http://www.graph500.org), a network and data intensive benchmark.

On such a large machine, parallel applications face several challenges to scale, and communication performance can be a major limiting factor. This paper covers a diversity of techniques showing how communication performance can be optimized using both hardware and software techniques developed through a coordinated co-design effort.

We first provide a detailed look at the performance of the BG/Q interconnection network on a number of important communication patterns. In particular, BG/Q provides multiple, flexible, and programmable hardware dynamic routing algorithms which support a diverse application set. We explore the routing algorithms' effectiveness for all-to-all, challenging bisection pairings, and random communications patterns. We also investigate how several software techniques can optimize and improve communications intensive benchmarks and applications. We describe optimizations, including multithreading and message aggregation, for the HPCC Random Access benchmark (http://www.hpcchallenge.org). While not an official HPCC submission, this paper reports how a 16 rack (16384 node) BG/Q achieves 858 Giga Updates per Second (GUPS), or 54 GUPS per rack. We also present results showing how the Algebraic Multigrid (AMG) application [9] and an iterative Poisson's equation solver can be accelerated using "communication threads" in which otherwise idle threads are used to offload and manage communications activity.

Our paper makes the following contributions:

- We demonstrate excellent performance achieved by the 5D BG/Q torus network for several all-to-all and bisection communication patterns.
- We develop a hybrid routing algorithm and show its effectiveness under non-uniform traffic loads.
- We show how the BG/Q system performance can be significantly improved by offloading communication activity to separate threads.
- We describe how the BG/Q messaging layer incorporates configurable features of the network, providing very good performance to the average user while still permitting the experienced user to select routing algorithms and messaging settings to further optimize application performance.
- We demonstrate excellent GUPS performance with a software-optimized version of the Random Access benchmark.

Taken as a whole, this paper shows the benefits of providing multiple hardware routing algorithms to more efficiently support different communication patterns. Furthermore, tight coordination between hardware and software can significantly accelerate communications. Offloading to software can in some cases reduce hardware complexity as will be illustrated in the paper.

II. Summary of BG/Q Network Architecture

To properly understand the results in this paper, we summarize the most relevent features of the BG/Q interconnection network architecture. For user applications, BG/Q presents a 5D torus with each link running at 2 GB/s (2 GB/s send + 2 GB/s receive). A subset of compute nodes, called bridge nodes, use an 11th link that attaches to BG/Q IO nodes. Including packet and protocol overhead, up to 90% of the raw data rate (1.8 GB/s) is available for user data. The network supports point-to-point messages, collectives and barriers/global interrupts over the same physical torus (BG/L and BG/P had separate networks for collectives and barriers). The machine can be partitioned into non-overlapping rectangular sub-machines. These sub-machines do not interfere with each other, except possibly on the IO nodes and its corresponding storage system. For point-to-point messages, BG/Q supports both determinsitic and dynamic routing with deadlocks being prevented via "Bubble" routing [10] in which packets can switch from a dynamic virtual channel to the bubble (deterministic) escape virtual channel when network tokens are exhausted. The deterministic routing is (programmably) dimension ordered; we have found that ordering the dimensions from longest first to shortest last is

typically best for performance. With this, queues for packets waiting to enter the bottleneck (longest dimension) links are actually stored in the memory system rather than in the much more limited network FIFOs.

Dynamic routing is also programmable enabling different routing algorithms to be used, on a per message basis, at the same time, i.e., a given message always uses the same algorithm but different messages can use different algorithms. This is called "zone routing" and implements in hardware ideas first explored in software on BG/L [11]. When a packet enters the network, it is assigned a vector of "hint bits," one bit per direction indicating whether the packet should move in the plus or minus direction for each dimension, until it reaches its destination. The hint bits may be assigned by hardware for minimal path routing or can be programmed by software. On BG/L, at each hop in the network, a packet may dynamically move in any direction for which a hint bit is specified. On BG/Q, a packet header also contains two bits which specify one of four ***zone IDs*** and the allowable movement of dynamic packets is constrained by programmable mask registers for each of the zone IDs. For example, the masks for one zone ID can be set so that packets must complete all hops in the longest dimension(s) first before moving to smaller dimensions, while for a different zone ID the masks could permit movement along any valid direction, as on BG/L. Each such mask is referred to as a ***zone***, and we refer to a specific mask as ***zone x of zone ID y***. To describe a zone ID, we use the following notation and example: {A}{BCD}{E}. This means that a packet first must travel to its final destination along the A dimension; then it may travel along the B, C, and D dimensions, taking hops in any order until all three of these dimensions are complete; and finally the packet routes along the E dimension until it reaches its final destination. Table I shows the zone routing masks which we use in this paper for selected system sizes. Experiments in [11] and near cycle-accurate simulations of the BG/Q network indicate that longest dimension(s) first to shortest dimension(s) last typically performs well. Conversely, we found that typically a shortest-to-longest approach did not perform well, so we do not include results here. Studies in this paper show that other, more flexible, forms of zone routing can be beneficial.

Note that in Table I, zone ID 3 is the same as the deterministic ordered zone ID 2 except that hops in dimension E are also permitted to occur first. In other words, packets are first injected and may switch between either the longest dimension in the system or dimension E. This can improve performance since the length of E is always 2, no packet can travel more than one hop in E. Even if the E network FIFOs are full of dynamic packets, they cannot block packets from longer dimensions turning onto E since those packets can use the bubble escape virtual channel. In this case the small additional contention from packets turning from E to the longest dimension may be outweighed by the additional buffering effect of allowing packets to inject into either dimension E or the longest dimension.

TABLE I: DYNAMIC ZONE ROUTING MASKS FOR SELECTED SYSTEM SIZES USED IN THIS PAPER.

Zone ID	Description	16 racks 16x8x8x8x2	4 racks 8x4x8x8x2	1 rack 4x4x4x8x2
0	Longest-to-shortest	{A}{BCD}{E}	{ACD}{B}{E}	{D}{ABC}{E}
1	Unrestricted	{ABCDE}	{ABCDE}	{ABCDE}
2	Deterministic ordering	{A}{B}{C}{D}{E}	{A}{C}{D}{B}{E}	{D}{A}{B}{C}{E}
3	Add E to the first zone of Det. ordering	{AE}{B}{C}{D}	{AE}{C}{D}{B}	{DE}{A}{B}{C}

To further improve performance, we explore the use of software "pacing" in which the fullness of packet queues within the network logic is controlled by limiting the injection rate of packets into the network, similar to TCP/IP window flow control. In our form of pacing, there is a window size of W bytes and each node is permitted to inject requests for at most 2W bytes at any one time. After W bytes are received, a remote get (rDMA read) request is issued for another W bytes (or the remaining message size).

The tests described in Sections III and IV are written using low level System Programming Interface (SPI) calls that access the network hardware resources directly [5], so as to eliminate most software overhead from the measurements. The GUPS results of Section V are obtained using the BG/Q production messaging library PAMI (Parallel Active Message Interface) [12]. PAMI uses SPI calls to access the hardware and supports both communication threads and a form of pacing. The BG/Q MPI implementation runs on top of PAMI.

III. ALL-TO-ALL BANDWIDTH

The peak all-to-all bandwidth (BW) of a torus is limited by the length of its longest dimension, since a given link in this dimension is utilized by more source-destination pairs. If the length of the longest dimension is L, then the peak user data per-node all-to-all BW is 8/L • 1.8 GB/s [11].

We ran our SPI-based large-message all-to-all performance test on systems up to 16 racks (16384 nodes). In this test, each node sends 32 KB of data to each of the other N-1 nodes. The data is broken up into a number of smaller messages of constant size which are sprayed randomly over the destinations. Breaking up the 32KB into smaller submessages had only a small effect since each node is already spraying packets from different messages throughout the network. To explore the effect of zone routing, we ran the test using dynamic routing zone IDs 0 through 3 as well as using deterministic routing on 4-rack and 16-rack systems, and the results are shown in Figure 2. The best results are achieved with zone ID 0, which is expected. Recall that in zone ID 0, packets are first routed along the longest dimension (here, A), which is the most heavily loaded in this case; so no packets turn onto A from other dimensions, mitigating the effect of contention. At the same time, once packets turn off of A, they turn onto less heavily loaded dimensions, so the effect of

multiple dimensions turning onto B, for example, is less severe than it otherwise would be. For 16K nodes, there is a single longest dimension of length 16, which is twice as long

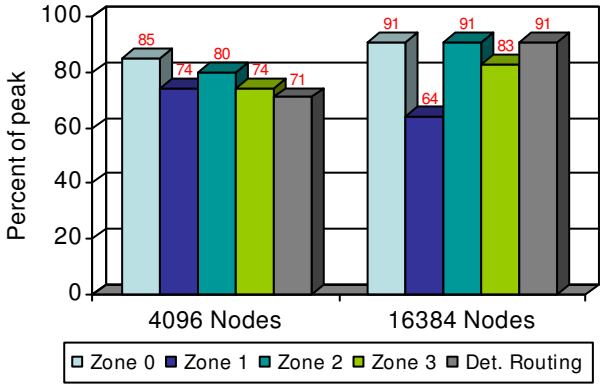

Figure 2: All-to-All Performance as a Percentage of Peak, for Dynamic and Deterministic Routing on 4 and 16-Rack Systems. Submessage size 4KB.

as the next longest dimensions. Since zone ID 2 and deterministic order also route the longest dimension first, their performance is similar to that of zone ID 0. On the more symmetric 4K nodes, with three longest dimensions of length 8, dynamic routing is able to more effectively distribute traffic throughout the network than deterministic routing.

We ran the all-to-all performance test on a wide range of system sizes, from 512 nodes up to 16384 nodes. All-to-all results for systems up to 2048 nodes were reported in [5] and are included along with the larger systems in Table II. Table II shows that as system size grows, the network is capable of sustaining excellent all-to-all bandwidth from 85% to 95% of peak using a longest-to-shortest dimension dynamic zone-routing approach. The PAMI implementation uses an algorithm that sprays traffic using zone ID 1 for systems 512 nodes and smaller, and it uses zone ID 0 for larger systems.

IV. Bisection Bandwidth

For a torus of N nodes with longest dimension of length L, the bisection bandwidth is (N/L) · 4 · B, where B is the bandwidth of a single unidirectional link.

A. *Diagonal and Furthest-Node Pairings*

One type of communication pattern which is useful for evaluating the effectiveness of an interconnection network at sustaining its bisection bandwidth is the *bisection pairing*. In a bisection pairing each node in the network communicates with exactly one other node, no two nodes communicate with the same node, and each source-destination pair crosses the bisection of the network exactly once. In this paper we evaluate two such challenging pairings, referred to as the diagonal and furthest-node pairings, as described below.

- Diagonal pairing: each node communicates with the node which is a reflection across the midpoint of each dimension. In each dimension the node with index i communicates with the node with index L-i-1, where L is the length of the dimension. On a mesh, these pairings are such that if you draw a line between each pair, they all pass through the center of the mesh.
- Furthest-node pairing: each node communicates with the node which is the maximum number of hops away.

We ran an SPI-level bisection performance test on 1-rack (1024 node) and 4-rack systems, using dynamic routing zone IDs 0-3 as well as deterministic routing, and the results are presented in Table III for diagonal pairing and Table IV for furthest-node pairing. We also vary the pacing of the message between nodes by changing the window submessage size. This has the beneficial effect of preventing the network from over-saturating and causing performance to deteriorate. Based on Tables III and IV, we observe that using a pacing window size of 8KB gives the best performance across all zone IDs, so throughout the rest of the paper we limit our results to this pacing window size. The bisection performance as a percentage of peak is significantly better on one rack than on four, especially for the more challenging diagonal pairing. This is due to the fact that there is a single long dimension in the one-rack system size, so that as discussed in Section III, packets are prevented from turning onto that long dimension. For more symmetrical system sizes with more than one long dimension, it is not possible to completely eliminate packets turning onto at least one of the long dimensions.

On the 4-rack system, the best routing for the diagonal pairing is zone ID 3, since it maintains high performance across a wide range of window sizes. For the furthest-node pairing, the best performance is achieved with zone ID 0 since this pairing naturally has a much more evenly distributed traffic pattern, equally utilizing all of the links, similar to the all-to-all case, so that the standard longest-to-shortest dynamic routing performs quite well. Conversely, the diagonal pairing does not evenly utilize the links, so that dynamic routing inadvertently concentrates the traffic on a relatively small number of links, including bisection links. By definition, in order to obtain a high percentage of the peak bisection bandwidth, all of the bisection links must be utilized. Deterministic (and deterministic-ordered dynamic) routing forces some of the traffic around the hot-spots and mitigates the congestion significantly.

A key observation is that there are some source-destination pairs in the diagonal pairing which have only one minimal path between them (i.e. a single hop in each dimension), and there are other pairs which have many possible paths between them. Of those paths, some overlap with the close pairs, and others avoid using the same links. We next explore whether it can be beneficial to use different zone IDs for different partners in order to diffuse the hot spots in the network.

B. *Flexibility Metric*

In order to differentiate between the pairs with varying numbers of minimal paths between them we introduce the

TABLE II: All-to-All Performance, as a Percentage of Peak for Zone ID 0 Dynamic Routing and 4 KB Submessage Size as a Function of System Size

# Nodes	512	1024	2048	4096	16384
Performance (%)	95	92	94	85	91

TABLE III: DIAGONAL PAIRING PERFORMANCE AS A PERCENTAGE OF PEAK BISECTION BANDWIDTH FOR 1 AND 4-RACK SYSTEMS USING VARIOUS ROUTING SCHEMES.

	1024 Nodes				
Window size (KB)	Zone ID 0	Zone ID 1	Zone ID 2	Zone ID 3	Det.
2	84	91	87	85	83
4	87	95	92	90	92
8	93	96	97	92	96
16	95	96	98	94	98
32	96	95	100	97	98
64	94	96	100	97	100
128	90	96	100	97	100
	4096 Nodes				
2	48	54	71	71	74
4	52	56	74	77	83
8	55	60	78	82	72
16	58	66	74	83	40
32	57	66	65	78	38
64	53	67	54	75	38
128	50	67	47	76	38

TABLE IV: FURTHEST-NODE PAIRING PERFORMANCE AS A PERCENTAGE OF PEAK BISECTION BANDWIDTH FOR 1 AND 4-RACK SYSTEMS USING VARIOUS ROUTING SCHEMES

	1024 Nodes				
Window size (KB)	Zone ID 0	Zone ID 1	Zone ID 2	Zone ID 3	Det.
2	86	71	90	72	88
4	92	83	92	77	90
8	96	85	97	80	96
16	98	78	98	77	98
32	98	72	98	78	98
64	97	73	98	80	98
128	88	72	98	83	98
	4096 Nodes				
2	70	61	62	58	65
4	77	76	68	63	76
8	92	83	72	66	76
16	93	84	68	65	44
32	90	75	63	70	23
64	83	72	55	68	22
128	76	75	50	60	22

flexibility metric:

$$F = \sum_{i=0}^{D-1} h_i / (L_i / 2),$$

where h_i is the number of hops in dimension i for the given source-destination pair; $L_i/2$ is half the length of dimension i (i.e. the maximum number of hops in a torus using minimal path routing); and D is the number of dimensions in the network. In our implementation dimension E is length 2 for all system sizes and thus can be ignored. Since $h_{max} = L_i/2$, $F_{max} = D = 4$ in this case. Furthermore, all traffic for the furthest-node pairing has $F = F_{max}$, since each message in that pairing travels the maximum distance in the torus. In general, there are a relatively small number of possible values of F for a given size system and communication pattern.

On a system size of 4 racks, the size of the network is 8x4x8x8x2. For the diagonal pairing on a torus, each packet takes an odd number of hops in each dimension. So on a dimension of length 4, all packets travel exactly 1 hop; on a dimension of length 8, either 1 or 3 hops. This means that the value of F for a dimension of length 4 is 0.5, and the two possible values of F for a dimension of length 8 are 0.25 and 0.75. So for this configuration, there are four possible sums of F for the diagonal pairing: 1.25, 1.75, 2.25, and 2.75. Our scheme uses two thresholds, T_h and T_l, to choose between zone IDs. For source-destination pairs with $F < T_l$ or $F >= T_h$, zone ID 0 is used; if $T_l <= F < T_h$, then zone ID 3 is used, since these are the zone IDs which give the best performance for the furthest-node and diagonal pairings respectively. Table V shows the performance results for a 4-rack system for the diagonal pairing with all possible combinations of T_h and T_l. The performance peaks at 85%, whereas the best performance with zone IDs 0 and 3 alone was 58% and 83% respectively (from Table III). So there is a constructive interference effect that keeps some traffic away from hot spots and mitigates the bottleneck effect.

TABLE V: PERCENTAGE OF PEAK BISECTION FOR SOME COMBINATIONS OF FLEXIBILITY METRIC THRESHOLDS FOR DIAGONAL PAIRING ON 4096 NODES, CHOOSING BETWEEN ZONE IDS 0 AND 3. PACING WINDOW 8KB.

T_l, T_h	1.5, 2.0	1.5, 2.5	1.5, 3.5	2.0, 2.5	2.0, 3.5	2.5, 3.5
Performance	63	85	85	63	66	60

Interestingly, the performance is best when only the smallest bucket (F = 1.25, closest pairs) is sent with zone ID 0 and either the rest, or all but the largest bucket (F = 2.75), are sent with zone ID 3. The smallest bucket represents messages which are travelling one hop in each dimension, so there are not enough routing paths to avoid hot spots and thus you want the benefit of sending longest to shortest dimension. Packets with a moderate amount of flexibility benefit by being forced to route in one dimension at a time. Packets with a very high degree of flexibility can be efficiently handled by the network's inherent dynamic routing mechanism. Performance is more sensitive to the value of T_l, the cutoff between low and moderate flexibility. For example, from Table V, holding T_h constant at 2.5 and increasing T_l from 1.5 to 2.0, performance drops from 85 to 63% of peak; whereas performance is unchanged when T_l is held constant at 1.5 and T_h increases from 2.5 to 3.5.

It is important to investigate this approach at a larger scale. The largest system size we have available at this time is 16 racks, or 16384 nodes. At this size, there are ten different values of F for diagonal pairing. Tables VI and VII show results for both the diagonal and furthest-node pairing communications with zone IDs 0-3, and deterministic routing, as well as selected flexibility metric thresholds combining zone IDs 0 and 3 for the diagonal pairing. First, as expected, the performance is better than on 4 racks since there is a single long dimension. For example, zone ID 3 on 16 racks achieves 91% of peak (Table VI) for the diagonal pairing whereas on 4 racks its best is 83% of peak (Table III). Using the flexibility metric for the diagonal pairing, up to 93% of peak (Table VII) is achieved on 16 racks compared to 85% on 4 racks (Table V).

TABLE VI: PERCENTAGE OF PEAK BISECTION, FOR SINGLE-ZONE ID ROUTING FOR DIAGONAL AND FURTHEST-NODE PAIRING ON 16384 NODES. PACING WINDOW 8KB.

Zone ID	0	1	2	3	Det.
Diagonal	71	62	77	91	85
Furthest-node	95	83	93	76	92

TABLE VII: PERCENTAGE OF PEAK BISECTION, FOR SELECTED COMBINATIONS OF FLEXIBILITY METRIC THRESHOLDS FOR DIAGONAL PAIRING ON 16384 NODES. PACING WINDOW 8KB.

T_l, T_h	1.0, 1.5	1.0, 1.75	1.0, 2.0	1.0, 2.25	1.0, 2.5	1.0, 2.75	1.0, 3.0
Performance	87	85	91	92	93	92	92
T_l, T_h	1.25, 1.5	1.25, 1.75	1.25, 2.0	1.25, 2.25	1.25, 2.5	1.25, 2.75	1.25, 3.0
Performance	85	85	94	92	94	93	93
T_l, T_h				1.5, 2.25	1.5, 2.5	1.5, 2.75	1.5, 3.0
Performance				72	72	72	72

As on 4 racks, the 16-rack performance is much more sensitive to the value of T_l than T_h. Trends seen in the smaller systems also apply to the larger 16-rack system. Routing messages with very low flexibility or very high flexibility with a longest-to-shortest zoned approach, while routing other messages with intermediate flexibility with a deterministic-ordering approach can provide better performance than either approach alone.

A form of pacing with the flexibility metric has been implemented in PAMI, and is thus used by MPI. Pacing is controlled by a thread on the seventeenth core and the flexibility metric thresholds are chosen differently depending on the system size. Default settings can be overridden using environment variables, permitting users to tune and optimize their codes.

C. *Random Pairing*

An important benchmark to evaluate the performance of an interconnection network is the random-pairing benchmark. In this benchmark, each node is randomly paired with another node in the system. Each node in the network communicates with exactly one other node; no two nodes communicate with the same node. As with all-to-all the expected per-node peak bandwidth is 8/L • 1.8 GB/s.

(s,k)-random pairing benchmark: Since the pairs are determined randomly and the aforementioned calculation only yields the peak bandwidth in expectation, it only serves as an upper bound. There can be local hot-spots due to the randomness of the pair selections, and this smoothes out as the number of pairs increases and eventually approaches a true all-to-all communication pattern. Thus, in order to get a better idea of the performance, we extend this benchmark as follows. We define an (s,k)-random pairing wherein each node utilizes s cores and each core communicates with k random partners on different nodes. Thus every node communicates with sk other nodes. Note that the *(1,1)-random-pairing* benchmark is equivalent to the *random-pairing* benchmark. The expected peak data-per-node BW is the same as before, i.e., 8/L • 1.8 GB/s.

TABLE VIII: RANDOM-PAIRING PERFORMANCE AS A PERCENTAGE OF PEAK FOR 1 AND 4-RACK SYSTEMS WITH DIFFERENT ROUTING SCHEMES WITH S=16 AND K=16. PACING WINDOW 8KB.

Number of Nodes	Zone ID 0	Zone ID 1	Zone ID 2	Zone ID 3	Det.
1024	56	67	54	57	51
4096	70	77	45	47	38

TABLE IX: RANDOM-PAIRING PERFORMANCE AS A PERCENTAGE OF PEAK BANDWIDTH FOR 1 AND 4-RACK SYSTEMS USING ZONE ID 1 ROUTING WITH S=16. PACING WINDOW 8KB.

Number of Nodes	k=1	k=2	k=4	k=8	k=16
1024	50	57	58	64	67
4096	65	66	72	75	77

We ran our SPI-based random-pairing tests on systems of 1 rack and 4 racks. In this test, we exchanged 1 MB of data between each pair in the (s,k)-random pairing. To explore the effect of zone routing, we ran the test using dynamic routing zone IDs 0 through 3 as well as using deterministic routing. These numbers are presented in Table VIII for s=16 and k=16. All the tests were performed using pacing with a window size of 8KB. We observe that the best results were obtained with zone ID 1 routing. We believe that local hotspots are more easily avoided using the unrestricted dynamic routing of zone ID 1 compared to the longest-to-shortest routing of zone ID 0. We also ran the tests with s=16 and k=1,2,4,8,16 in order to study the effect of increasing communication partners on the performance. The results are shown in Table IX; these were obtained with zone ID 1 routing and with pacing. As expected, performance steadily improves as the number of communication partners increases. With (s,k) = (16,16), performance goes as high as 77% on 4096 nodes. We also see that performance on 4096 nodes is significantly better than on 1024 nodes. On the larger system, this is probably due to the more symmetric topology, more opportunity for dynamic routing to avoid hotspots, and a smaller likelihood of selecting adversarial pairings such as multiple collinear pairs.

D. *Reverse*

The reverse benchmark evaluates the performance of the interconnection network at sustaining bisection bandwidth on an irregular communication pattern. In this benchmark, a node with MPI rank X communicates with the node having rank Y where the bit representation of the coordinates of Y on each dimension are obtained by reversing the bit pattern of the corresponding co-ordinate of X, i.e., for any dimension A and bit i (i = 0, 1, …, $\log_2 L_A - 1$), the i^{th} bit of Y along dimension A is the same as $(\log_2 L_A - i - 1)^{th}$ bit of X along dimension A, where L_A is the length of the dimension A. The peak performance for this benchmark is calculated by examining the central cut along the longest dimension. For 4 racks

TABLE X: REVERSE PERFORMANCE AS A PERCENTAGE OF PEAK FOR 1 AND 4-RACK SYSTEMS WITH DIFFERENT ROUTING SCHEMES. PACING WINDOW 8KB.

Number of Nodes	Zone ID 0	Zone ID 1	Zone ID 2	Zone ID 3	Det.
1024	94	93	94	94	94
4096	65	65	75	75	53

TABLE XI: TRANSPOSE PAIRING LOAD ON TORUS NETWORK LINKS WITH STATIC ROUTING.

Nodes	Routing Dimension Order	Rank to Coord Mapping	Max link load	Predicted % of bisection throughput
1024	DBCAE	ABCDE	4	50 %
1024	DBCAE	CDABE	4	50%
4096	ADCBE	ABCDE	16	12.5%

(8x4x8x8x2), the longest dimension is of size 8, which is represented by 3 bits. The node pairs that communicate with each other are the pair [1 (001), 4 (100)] and the pair [3 (011), 6 (110)]. Note that both of these communicating pairs use the link between nodes 3 and 4 (they do not use the diametrically opposite link of the torus). Thus when we look at the cut across the longest dimension, the total amount of data passing though the cut is twice the data generated on each node. Therefore the peak data-per-node BW is 2 • 1.8 GB/s. Similarly, for 1 rack (4x4x4x8x2), the longest dimension is 8, and hence the peak data-per-node BW is again 2 • 1.8 GB/s.

We ran our SPI-based reverse-pairing tests on systems of 1 rack and 4 racks. In this test, we exchanged 1 MB of data between the communicating pairs. To explore the effect of zone routing, we ran the test using dynamic routing zone IDs 0 through 3 as well as using deterministic routing. The results are shown in Table X. We observe that on 4096 nodes, the performance with dynamic routing zone IDs 2 or 3 is approximately 75% of the peak. The performance of the flexibility metric approach is between that of zone IDs 0 and 3, as expected. On 1024 nodes, the performance is very consistent across the different zone routings and reaches 95% of the peak.

E. Transpose

In transpose benchmark, the nodes on the network form a virtual 2D square matrix where each node (x,y) is paired with the node (y,x). Diagonal nodes (x,x) do not participate in this pairing communication operation. On the 5D BG/Q torus network, the 2D mesh is overlaid on the dimensions of the 5D torus. Depending on how the processes are mapped to the dimensions of the 5D torus, it may be possible to fold the dimensions of the 5D torus to form a 2D mesh. For example, on 1024 nodes when dimensions A,B,C,D,E have sizes 4x4x4x8x2 respectively, a 32x32 virtual mesh can be formed as {CD}x{ABE} when CDABE mapping is used. Other mappings such as ABCDE may result in a dimension (C) being shared by both X and Y dimensions in the mesh.

As shown in [15], the transpose pairing is a challenging communication operation that can cause hotspots along the diagonal nodes in the 2D Mesh. On a 5D torus with static deterministic routing packets will converge towards the hotspot diagonal nodes resulting in lower overall throughput. We developed a simple program to compute the load on the links for the transpose communication pattern with deterministic routing and the results are presented in Table XI.

TABLE XII: TRANSPOSE PERFORMANCE WITH DETERMINISTIC AND ADAPTIVE ROUTING.

Nodes	Rank to Coord Mapping	% of Bisection Throughput for Adaptive Routing zone ID 1	% of Bisection Throughput for Deterministic Routing
1024	ABCDE	83%	31%
1024	CDABE	89%	41%
4096	ABCDE	74%	13%

Observe that with deterministic routing, links around the hotpots have several messages passing through them and the achievable percent of peak is quite small. With adaptive routing, where the torus routers send packets along the least loaded links, significant improvement in performance can be expected. Table XII shows the percent of bisection throughput achieved with dynamic routing on zone ID 1 and deterministic routing for the transpose operation using a pairing test written in SPI. The percent is adjusted to account for the fact that only N - sqrt(N) nodes participate in the transpose operation. Observe adaptive routing with zone ID 1 achieves higher throughput than deterministic routing as it can smoothen network load around hotspots. We also observed better performance when the 5D torus can be folded to form the 2D mesh. Note that in Table XII, mapping CDABE performs better than ABCDE. Zone ID 1 performs best as it has the most flexibility in moving packets around hotspots. Other zone IDs 0,2,3 achieve throughputs between deterministic routing and zone ID 1, as does the flexibility metric approach.

V. GUPS

A. Introduction

Random access performance of the memory subsystem is critical to many applications. The HPCC suite includes the Random Access benchmark which measures the capability of a system to generate and apply updates to random locations in the memory. On earlier machines, Blue Gene/L and Blue Gene/P [13], 3D bucketing algorithms have been designed to amortize the transfer costs by aggregating multiple updates into a single bucket. Such techniques lower the software costs of injection and reception of the update and also help in better utilization of the network. The performance of the benchmark is measured in GUPS and is bounded by the bisection performance of the network, although other factors such as software overhead could be the bottleneck. Further, the total amount of look-ahead depth for aggregation is restricted to 1024 updates per process or 8192 bytes with eight bytes per update, limiting the size of the buckets used.

B. GUPS design on BG/Q

The benchmark is run with sixteen processes per node, one process per core, with each process utilizing four threads. Out of the four threads, two threads are completely dedicated for software routing and the other two are used for generating the

updates and applying the updates. The salient features of the new design are the following.

1) Software routing for the five dimensional torus: Because BG/Q has multiple threads per core, there exist new bucketing opportunities. In addition, the 5D torus permits larger buckets compared to a 3D torus with the same number of nodes. For example, in a 64K node 64x32x32 3D system the process handling the longest dimension has 64 buckets whereas a 16x16x16x8x2 5D system has at most 16 buckets per dimension.

In the design proposed in this paper, a process is required to route traffic from only one incoming dimension to only one outgoing dimension. This greatly reduces the number of buckets thus allowing for more aggregation. For example, on the largest machine, the number of send buckets utilized would only be around 16. The basic idea is to aggregate all the updates from the processes on a node and then route them along the dimensions of the torus. Once the updates reach the final destination node, they are scattered to their respective processes. Also, the packets are always routed from the shorter to the longer dimensions to increase message aggregation and to avoid any cyclic dependencies. In a 16 rack system, the E dimension is the shortest and the A dimension is the longest.

2) Translating communication parallelism into GUPS performance: The MU provides a high level of parallelism within a node with multiple injection and reception FIFOs operating concurrently on different messages. For example, within a single process, multiple threads can send and receive messages on separate hardware FIFOs eliminating the need for shared locks. PAMI on BG/Q exposes this concurrency in the form of higher level abstraction such as contexts. Further, these threads can be pinned to a specific context and are addressed using end-points. A complete discussion on these concepts is given in [12]. Our design of GUPS uses these PAMI concepts as building blocks and the entire algorithm is implemented in the pre-registered message handlers.

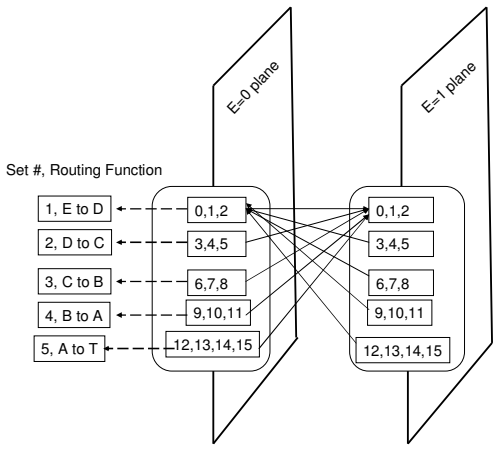

Figure 3. Routing along E planes.

Our new design harnesses communication parallelism by allowing threads in more than one process to route in the same dimension. Processes belonging to one *routing set* drain packets from the reception FIFOs of a lower dimension and route to the *routing set* of processes of a higher dimension. Further, each process spawns two independent routing threads working in parallel, for a total of 32 routing threads per node.

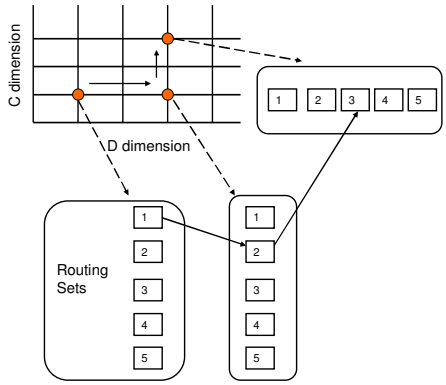

Figure 4. Dimension ordered routing in the routing sets.

3) Detailed illustration of the parallel software routing: The initial routing step is explained as follows. As shown in Figure 3, the sixteen processes on a node are divided into five routing sets. All these processes, after generating the updates, route to routing set 1, comprised of processes with local ranks {0, 1, 2}. The other routing sets numbered from two to five are also shown in Figure 3. As explained below these are used for routing along the remaining dimensions of the network, D to A. The T dimension is the local dimension, and processes in routing set 5 with local ranks {12, 13, 14, 15} are used in the last step of the software routing and forward the updates to all sixteen processes within the node. Note that only the first thread of these processes is used to generate the updates. Apart from generating the updates, the thread also maintains two buckets, corresponding to the E = 0 and E = 1 plane. All the updates are aggregated into these buckets before sending to the processes of routing set 1. As shown in Figure 3, processes in the E = 0 plane communicate with routing set 1 of E = 1 plane via the network. For communicating to the processes in the same plane, the updates utilize shared memory. Note that in the initial phase of the algorithm, thread 0 of each process communicates to threads 1 and 2 of the processes belonging to routing set 1 in order to aggregate all the updates on a node. By careful mapping, we allow for uniform distribution of updates to each of the routing threads belonging to the three processes of a routing set.

The remaining routing steps traverse the dimensions of the network in the order D→C→B→A. A further optimization to generalize the algorithm for any arbitrary system configuration would be to go from shortest to longest dimension to get the most aggregation of the updates. However, it is to be noted that

on a complete 96 rack machine, the ordering required is the same as in this paper. Figure 4 shows two hops of this routing first along the D dimension followed by the C dimension. As indicated in Figure 4, the packets injected by routing set 1 are received by the processes belonging to the routing set 2. For the C dimension, updates travel from routing set 2 to 3.

C. *Peformance evaluation:*

The performance of the Random Access benchmark is tightly coupled to the bucket size used for message aggregation. In the following we describe the calculation used to obtain the bucket sizes. We first enumerate the types of buckets used per process in our design:

1) Issue Send buckets: Used by the issue thread 0 which generate the random numbers and send updates along the E dimension. There are two issue send buckets one for each E plane.

2) Routing Send buckets: Used by the routing threads 1 and 2 to send along a given dimension. The number of routing send buckets is the same as the dimension size.

3) Routing Receive buckets: Used by routing threads 1 and 2 to receive updates. There is one routing receive bucket to process data received in the active message handler.

4) Final Update Receive buckets: There is one final update receive bucket that is used by the update thread 3 to receive the final updates.

An issue send bucket size of 512 B was experimentally determined to maximize performance. Similarly, the final update receive bucket size was experimentally selected at 256 B. The benchmark allows 8 KB of total bucket memory space, thus the remaining space for the routing send and receive buckets is (8192 - 512 - 256) = 7424 B, or 3712 B for each of the two routing threads. A routing send bucket is required per node along a dimension, as well as a single receive bucket. Thus each routing send and receive bucket is 3712/(dimension_size + 1) bytes as there are dimension_size sending buckets and one receive bucket.

Since GUPS follows an all-to-all kind of pattern and there are 8 bytes per update, the network bound on updates per second per node is ((8/L)*B)/8 = B/L, where B is the peak link bandwidth obtained after adjusting for the per packet overhead used in the software routing. For example, for a packet size of S bytes, B = S/(S+52)*2.0 GB/s, where 52 is the total number of bytes used in the header, trailer and the ack of the packet. S is determined from the bucket sizes used. From 1 to 8 racks, the network bound is over 200 million updates per node per second, and it is 100 million updates per node per second for 16 racks up to the full system size. From experimental evaluation, we observed that the performance achieved on a single node is 106 Million updates per second. Since each update requires a read and write of 128 B, this corresponds to an off-chip memory bandwidth of 27.6 GB/s. We use 106 million updates per second as the memory system hardware limit.

Table XIII reports the total GUPS, the update rate per node, and the hardware bound per node which is the minimum of the network and memory bounds for system sizes of 1 to 16 racks. For 16 racks, we achieved 858.1 GUPS which is 52.4% of the hardware bound. In our experience, the bottleneck for obtaining a higher GUPS performance is the processing costs for software routing on the five network dimensions and one local dimension.

VI. Optimization of applications Using Communication Threads

Each core of the BG/Q node has four hardware threads that share the core's resources. In applications that have high communication overheads, one of these threads can be dedicated to accelerate communication. We observed that in some hybrid applications that use OpenMP within the node and MPI to communicate across nodes, best performance is achieved with two or three hardware threads per core for computation. This is because OpenMP overheads may cancel the benefit of the additional threads for computation. MPI libraries on Blue Gene/Q can enable one or two communication threads per core to optimize the above mentioned scenarios. In addition to improving the overall messaging performance, communication threads also enable independent progress in the messaging stack that can be highly advantageous for asynchronous communication.

Past work [12] shows that we achieve a message rate of 107 million messages per second via the PAMI API and 20.9 million messages per second via MPI using 32 processes per BG/Q node. MPI has higher overheads than PAMI, as messages have to be matched on the receiver, based on the tag and the source rank. MPI libraries enable communication threads to accelerate the message processing, so that applications can take advantage of the high message rate available in the BG/Q torus network. With 8 processes per node, MPI libraries achieve a message rate of ~10 million messages per second with communication threads, and ~8 million messages per second without communication threads. The difference with or without communication threads is more pronounced when there are fewer processes per node.

To demonstrate the benefits of communication threads we present case studies with two linear algebra applications, Algebraic Multi-Grid (AMG) and an iterative Poisson solver. Both these applications are weak scaling applications where the problem size is increased proportionate to the increase in the number of cores. They also send and receive several messages of different sizes in each iteration.

The AMG method is used to iteratively solve partial differential equations using a heirarchy of grids with different resolutions. The communication pattern is dense near neighbor where processes send and receive hundreds of messages in each iteration of the solver, as processes having coarse grid points must communicate with a number of processes that have fine grid points. The size of the messages can vary from a few bytes to several hundred KB. To achieve high throughputs AMG requires the messaging libraries to achieve high message rates. On BG/Q we achieve the high messaging rates by enabling communication threads. We ran the AMG benchmark from the Sequoia Benchmark suite [9] with refinement levels of 8x8x8 using "solver 3", which uses a preconditioned generalized minimum-residual iterative method. We measured performance with and without communication threads. Table XIV presents the application throughput computed as a Figure Of Merit (FOM = system_size * iterations / iteration_time). These measurements used four MPI processes per node, and three threads per core for a total of 12 OpenMP threads per process, leaving one hardware thread per core available for communication threads. Note, even without communication threads, the best performance achieved in AMG is with three threads per core (i.e. exchanging the communication thread for an additional OpenMP thread does not improve performance), possibly because OpenMP overheads cancel the gains from an additional SMT thread. For example, on 512 nodes the FOM achieved with both 3 and 4 threads per core is 1.38e9. The performance improvement in the overall solver time due to communication threads is between 3.3 and 6.2%.

A more dramatic improvement was observed with a simple iterative solver for Poisson's equation. This solver was used to represent applications where the main communication pattern is boundary exchange on a regular grid. The computational performance of this solver is limited by bandwidth to memory. As a result, the overall performance of the solver is optimized by using two or three threads per core for computation, leaving one or two threads per core available for communication. Table XV shows the benefit of using communication threads with this simple iterative solver. Note that it is not meaningful to compare the step times between differently sized systems, since the number of iterations changes.

VII. Related Messaging Stack (PAMI) Features

The SPI-level research on message pacing, the flexibility metric, and commthreads described in this paper have been incorporated into the messaging stack (MPI / PAMI) as described below. Similar performance results are observed.

Message pacing is controlled by an "agent" thread that runs on the 17th core of each node. PAMI posts a given message to the agent for pacing when the size of the block exceeds 1 rack, and the message is larger than a W bytes (default 64KB), and the destination node is more than H hops away (default 4) or its ABCD coordinates differ from the source node's in more than D dimensions (default 1). The agent multiplexes the messages posted from the processes on the node and paces them, controlling the amount of data in the network on behalf of the node. The agent divides each message into windows of size W and allows up to M simultaneous windows in the network. The defaults for W and M vary based upon the block size and are currently empirically determined but can be overridden at job launch time by the user. The agent round robins through its list of messages, injecting one window from each message, pausing as needed to wait for a previous window to finish in the network, maintaining up to M active windows in the network. Software controlled pacing leverages the many threads on the BG/Q system and provides more flexibility versus the added complexity of a hardware-based pacing implemention.

TABLE XIII: Random Access Performance

Nodes	Total GUPS	Million Updates per Node per Second	Hardware Bound for Millions Updates per Node per Second
1024	47.3	46.2	106
2048	95.2	46.5	106
4096	184.7	45.1	106
8192	485.9	59.3	106
16384	858.1	52.4	100

In PAMI, the flexibility metric is used to determine the network routing for point-to-point messages that exceed F bytes (default 64KB). Routing can be deterministic, or dynamic with zone ID 0, 1, 2, or 3. One of two routing methods is used, depending on whether the metric between the source and destination nodes is within the range or outside of the range. The metric determines the routing for both paced and non-paced messages. Messages F bytes or less are deterministically routed by default. The default metric range, routing values, and threshold F vary based upon the block size and can be overridden by environment variables. The default configuration gives good performance for all of the bisection pairings, with the exception of the extreme transpose pairing. PAMI allows the user to override the default zone ID used at job launch time with environment variables; or the user can specify what zone ID to use on a message-by-message basis through the use of SPI calls. Modifying the zone IDs themselves (e.g. the dimension ordering) is possible but requires system calls. Ongoing research is in progress to refine this approach for when there are a large number of outstanding communication partners.

PAMI has contexts and commthreads that enable parallel communications progress. Contexts are a method of dividing messaging hardware resources so that parallel operations can occur. Commthreads run on hardware threads not being used by the application. The number of contexts and commthreads is determined by PAMI when MPI is initialized. Each context initially has its own commthread that makes progress on that context. If the application creates threads on the same hardware thread that is running a commthread, the

TABLE XIV: FIGURE OF MERIT (FOM) FOR THE AMG APPLICATION

Nodes	Process / node	OMP Threads / Process	FOM without Comm. Threads	FOM with Comm. Threads	% Gain
512	4	12	1.38 e+9	1.45 e+9	5.0
1024	4	12	2.42 e+9	2.57 e+9	6.2
2048	4	12	4.27 e+9	4.41 e+9	3.3

TABLE XV: STEP TIME (SECONDS) FOR THE POISSON SOLVER KERNEL

Nodes	Process / Node	OMP Threads / Process	Step Time (s) w/o comm threads	Step Time (s) with comm threads	% Gain
512	8	6	3.682	3.076	19.7
1024	8	6	2.525	2.258	11.8
2048	8	6	5.784	5.073	14.0

commthread gives its context to another commthread and yields to the application thread. Messages are associated with a contexts based on the destination rank and MPI communicator. This evenly distributes messages among the contexts while maintaining MPI ordering semantics. MPI posts a message to the contexts. The commthread picks it up, makes progress on it, and completes it. The main application thread finishes computing and finds the message completed. There are two versions of the MPI library on BG/Q. One is enabled for threaded operations and one is optimized for non-threaded operations. To have commthreads, the application must be linked with the thread-enabled MPI library and must initialize MPI with MPI_THREAD_MULTIPLE.

VIII. CONCLUSIONS

The Blue Gene/Q integrated network offers programmable "zone routing" control for dynamic (adaptive) routing. Using low level SPI programming and default zone settings, all-to-all performance ranges from 85% to 95% of the theoretical peak on 512 to 16K nodes. With 16K nodes, a software optimized version of the Random Access benchmark from the HPCC suite achieves a preliminary result of 858.1 GUPS. With this result, we learned that careful hardware/software co-design can lead to a thin but efficient software layer for message aggregation, thus changing a short-message random communication pattern into longer messages that perform well on the torus topology.

We studied the performance of difficult bisection pairings. Diagonal and furthest-node pairings each achieve good performance, albeit with different zone ID settings. Thus, a software-controlled flexibility metric routing mechanism is developed where different hardware routing algorithms are selected depending on the distance messages travel. Both the diagonal and furthest-node pairings achieve over 90% of peak on 16K nodes. This shows the importance of having multiple hardware routing options since different applications perform optimally under different routing algorithms. The flexibility metric enables the system to provide very good performance with default settings but still allows individual applications the opportunity to optimize further.

When running communication intensive applications using MPI, it is often beneficial to have dedicated MPI communication threads. The performance improvement ranges from 3.6% to 6.2% for AMG, and 11.8% to 19.7% for an iterative solver for Poisson's equation, on 512 to 2K nodes.

An initial implementation of the low-level PAMI API used by MPI automatically manages pacing, the flexibility metric, and communication threads. These settings can also be overridden by the end user. The Blue Gene/Q architecture provides the capability to fine-tune many hardware features. How higher level libraries such as MPI can best exploit these hardware features is an ongoing investigation.

ACKNOWLEDGMENTS

The Blue Gene project is a team effort. We would like to thank the entire IBM Blue Gene team for their contributions and support that made this work possible.

The Blue Gene/Q project has been supported and partially funded by Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the U.S. Department of Energy, under Lawrence Livermore National Laboratory subcontract no. B554331. We acknowledge the collaboration and support of Columbia University and the University of Edinburgh.

REFERENCES

[1] A. Gara, M. A. Blumrich, D. Chen, G. L,-T. Chiu, P. W. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the Blue Gene/L system architecture," *IBM Journal of Research and Development*, vol 49, no. 2/3, pp. 195–212, March/May 2005

[2] IBM Blue Gene Team, "Overview of the IBM BlueGene/P project," *IBM Journal of Research and Development*, vol. 52, no. 1/2, pp. 199-220, January/March 2008

[3] R.A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, P. A. Boyle, N. H. Christ, C. Kim, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M.A. Blumrich, R.W. Wisniewski, A. Gara, and G. L.-T. Chiu, "The IBM Blue Gene/Q Compute Chip", *IEEE Micro*, vol. 32, no. 2, pp. 48-60, (Mar/Apr 2012).

[4] D. Chen , N.A. Eisley, P. Heidelberger, R.M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D.L. Satterfield, B. Steinmacher-Burow, J.J. Parker, "The IBM Blue Gene/Q Interconnection Network and Message Unit," *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis (SC 11),* ACM Press, 2011, article 26.

[5] D. Chen, N.A. Eisley, P. Heidelberger, R.M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D.L. Satterfield, B. Steinmacher-Burow, J.J. Parker, "The IBM Blue Gene/Q Interconnection Fabric", *IEEE Micro*, vol. 32, no. 1, pp. 32-43, -(Jan/Feb 2012).

[6] S. Scott and G. Thorson, ''The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus,'' *Proceedings of HOT Interconnects IV*, August 1996, pp. 147–156.

[7] R. Alverson, D. Roweth, L. Kaplan, "The Gemini System Interconnect," *18th IEEE Symposium on High Performance Interconnects*, August 2010.

[8] Y. Ajima, Y. Takagi, T. Inoue, S. Hiramoto and T. Shimizu, "The Tofu Interconnect", *IEEE Micro*, vol. 32, no. 1, pp. 21-31, (Jan/Feb 2012).

[9] Sequoia Algebraic Multi Grid (AMG) benchmark https://asc.llnl.gov/sequoia/benchmarks/#amg

[10] V. Puente, R. Beivide, J. A. Gregorio, J. M. Prellezo, J. Duato, and C. Izu, ''Adaptive Bubble Router: A Design to Improve Performance in Torus Networks,'' *Proceedings of the IEEE 274 International Conference on Parallel Processing*, September 1999, pp. 58 – 67

[11] S. Kumar, Y. Sabharwal, R. Garg and P. Heidelberger, Optimization of All-to-all communication on the Blue Gene/L supercomputer, *In Proceedings of International Conference on Parallel Processing (ICPP)*, Portland, Oregon, 2008.

[12] S. Kumar, A.R. Mamidala, D.A. Faraj, B. Smith, M. Blocksome, B. Cernohous, D. Miller, J.Parker, J. Ratterman, P. Heidelberger, D. Chen, and B. Steinmacher-Burow. PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer. *To appear in proceedings of International Parallel and Distributed Symposium (IPDPS 12)*, Shanghai, China, May 2012

[13] V. Aggarwal, Y. Sabharwal, R. Garg, and P. Heidelberger,, "HPCC RandomAccess benchmark for next generation supercomputers*", IEEE International Symposium on Parallel & Distributed Processing, 2009 (IPDPS 2009). Pp. 1-11, 2009.*

[14] The Blue Gene Team, "Blue Gene/Q: by co-design," *to appear in International Supercomputing Conference,* June 2012.

[15] Fabrizio Petrini and Marco Vanneschi. Minimal vs. non Minimal Adaptive Routing on k-ary n-cubes. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96)*, Volume I, pages 505-516, Sunnyvale, CA, August 1996.

[16] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-driven, highly-scalable dragonfly topology. SIGARCH Comput. Archit. News, 36:7788, June 2008.

[17] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony. The PERCS High-Performance Interconnect. In 2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI), pages 75 82, August 2010.

[18] Steve Scott, Dennis Abts, John Kim, and William J. Dally. 2006. The BlackWidow High-Radix Clos Network. In *Proceedings of the 33rd annual international symposium on Computer Architecture* (ISCA '06). IEEE Computer Society, Washington, DC, USA, 16-28